Lecture 9

Interactive computer graphics

Menus, interactive modeling, animation

Menus

- Using mouse interaction and graphics primitives we can build our graphics input devices (widgets)
- Each specific window system provide a toolkit that contains ready to use input devices
- The cross-platform GLUT window system has some input devices (Widgets). Pop-up menu is one of them
- To use a pop-up menu in GLUT
  - We must define an action to each menu entry
  - We must link the menu to a particular mouse button
  - We must register a callback function for each menu

Menu example

- In a program that draw squares on the screen when the user click the left mouse bottom (the cursor must be on the graphics window)
- Menu entries
  - The first is to exit the program
  - The second is to make the drawn square larger
  - The third is to make the drawn square smaller
- The menu callback function name is demo_menu

Pop-up menu demo

```
/* globals */
GLsizei wh = 500, ww = 500; /* initial window size */
GLfloat size = 3.0;   /* half side length of square */
```

```
void myReshape(GLsizei w, GLsizei h)
{
    /* adjust clipping box */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, (GLdouble)w, 0.0, (GLdouble)h, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    /* adjust viewport and clear */
    glViewport(0,0,w,h);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
    /* set global size for use by drawing routine */
    ww = w;
    wh = h;
}
```

```
void myMouse(int btn, int state, int x, int y)
{
    if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN) drawSquare(x,y);
    if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)   exit(0);// the menu gain
}
void display()
{

}
```

```
void demo_menu(int id)
{
    switch(id)
    {
    case 1: exit(0);
        break;
    case 2: size =2 * size;
        break;
    case 3: if(size >1) size = size/2;
        break;
    }
    glutPostRedisplay( );// invoke display function to redraw without the menu
}
```

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Mouse event");
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutCreateMenu(demo_menu);
    glutAddMenuEntry("quit",1);
    glutAddMenuEntry("increase square size", 2);
    glutAddMenuEntry("decrease square size", 3);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

```
void drawSquare(int x, int y)
{
    y=wh-y;// convert from window coordinates to world coordinates
    // choose a random color
    glColor3ub( (char) rand()%256, (char) rand()%256, (char) rand()%256);
    glBegin(GL_POLYGON);
    glVertex2f(x+size, y+size);
    glVertex2f(x-size, y+size);
    glVertex2f(x-size, y-size);
    glVertex2f(x+size, y-size);
    glEnd();
    glFlush();
}
```

Picking

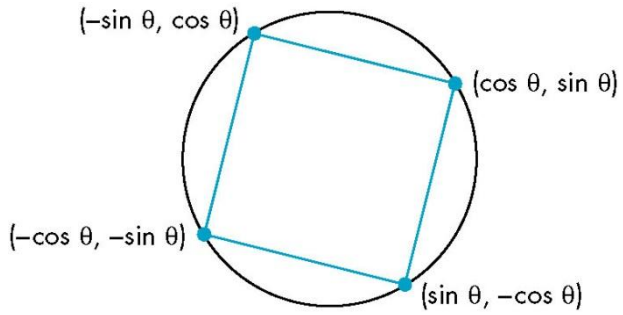
- The application program puts primitives in the GL pipelines to be processed and finally reach the frame buffer ad rasterized pixels
- We specify objects using primitives (for example, a sphere is approximated using a set of triangles at the poles and a set of quads in between the two poles
- Picking is the process of identifying a primitive or an object in your model by interacting with the contents of the graphics window on the screen (for example click to identify a sphere on the screen that contains many other, may be overlapping, objects
- OpenGL supports picking but requires some programming intervention (the details in sections 3.8,3.9,3.10 are optional)

Picking and interactive modeling

- In a graphic application that supports interactive modeling, the user, not the programmer, specify his model.
  - The user of the AutoCAD (civil or architect engineer) specifies the model (building)
- In an interactive modeling application
  - The user can add or remove parts of the model (walls, windows, doors, for examples)
  - The user can pick an object to delete, modify, move, etc.
  - The user can save/reload his model
  - The model must be maintained in a data structures to support interactive modeling

Animation: The rotating square

Consider the following two-dimensional point:
$x = \cos(\theta)$,
$y = \sin(\theta)$.
This point lies on a unit circle regardless of the value of θ. The three points
(- sin (θ), cos (θ)),
(- cos (θ), - sin (θ)),
(sin (θ), - cos (θ))
also lie on the unit circle. These four points are equidistant along the circumference of the circle



Unit circle

We can program an animating square by defining a global variable for (θ), The display function that displays a square defined by the above for points, and the idle callback function that increase (θ) by an increment each time invoked

Rotating square

```
float theta=30,thetar;
```

```
void wait(int timeout)
{
    timeout += clock();
    while(clock() < timeout)
        continue;
}
```

```
void idle()
{
    theta+=1; /* or some other amount */
    if(theta >= 360.0) theta-=360.0;
    wait(100);// control by the value according to system speed
    glutPostRedisplay();
}
void mouse(int button, int state, int x, int y)
{
    if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
        glutIdleFunc(idle);
    if(button==GLUT_MIDDLE_BUTTON && state==GLUT_DOWN)
        glutIdleFunc(NULL);
}
```

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    /* convert degrees to radians */
    thetar =theta/(3.14159/180.0);
    glVertex2f(cos(thetar), sin(thetar));
    glVertex2f(-sin(thetar), cos(thetar));
    glVertex2f(-cos(thetar),-sin(thetar));
    glVertex2f(sin(thetar),-cos(thetar));
    glEnd();
    glFlush();
}
```

```
void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
    glMatrixMode(GL_MODELVIEW);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glColor3f(0.0,0.0,0.0);
}
```

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Rotating square");
    glutDisplayFunc(display);
    myinit();
    glutMouseFunc(mouse);
    glutIdleFunc(idle);
    glutMainLoop();
}
```

Animation and flickering/artifacts

- As long as the contents of the frame buffer are unchanged and we refresh at the 60- to 85-Hz rate, we should not notice the refresh
- When drawing a simple image that takes less than one refresh cycle timer (the drawing starts and ends in the same cycle) the image should be smooth
- When drawing a complex image that takes longer than one refresh cycle time, we se different parts each refresh cycle. Hence, artifacts may occur
- When clearing and redrawing a simple object (square for example) there is no coupling between when new squares are drawn into the frame buffer and when the frame buffer is redisplayed by the hardware. Thus, depending on exactly when the frame buffer is displayed, only part of the square may be in this buffer. Hence, flickers and artifacts may appear

Reducing flickering and artifacts using double buffering

The objective is to display a scene only after finishing drawing it

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    /* convert degrees to radians */
    thetar =theta/(3.14159/180.0);
    glVertex2f(cos(thetar), sin(thetar));
    glVertex2f(-sin(thetar), cos(thetar));
    glVertex2f(-cos(thetar),-sin(thetar));
    glVertex2f(sin(thetar),-cos(thetar));
    glEnd();
    glFlush();
    if(doubleBuffering)glutSwapBuffers();
}
```

The drawing is done always in the back buffer when double buffering is used. Just swap buffers when finishing drawing

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    if(!doubleBuffering)glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    else glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Rotating square");
    glutDisplayFunc(display);
    myinit();
    glutMouseFunc(mouse);
    glutIdleFunc(idle);
    glutMainLoop();
}
```

Activate double buffering

Controlling the time of drawing/redrawing cycle in animation using timer

```
int n = 60; /* desired frame rate */
glutTimerFunc(100, myTimer, n);
```

In main function: Delay the main loop 100 milliseconds then call the my timer function

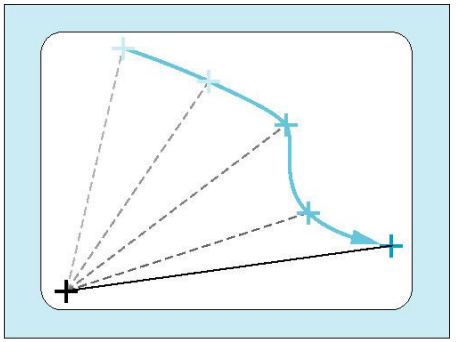
```
void myTimer(int v)
{
  glutPostRedisplay);
  glutTimerFunc(1000/n, myTimer, v);
}
```

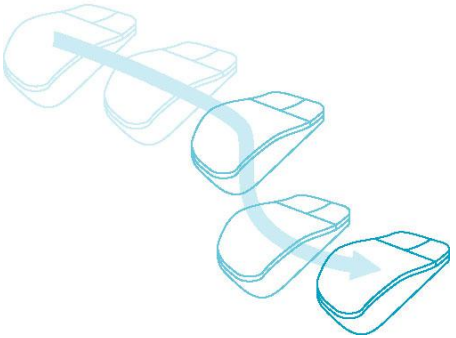
The callback: Recall the timer

Features of a good interactive program

- A smooth display, showing neither flicker nor any artifacts of the refresh process
- A variety of interactive devices on the display
- A variety of methods for entering and displaying information
- An easy-to-use interface that does not require substantial effort to learn
- Feedback t o the user
- Tolerance for user errors
- A design that incorporates consideration of both the visual and motor properties of the human

Interactivity and the need to work directly in the frame buffer





Displaying a popup menu requires drawing the menu on the screen, then restoring the screen contents it its place after it has been disappeared

In such cases we need to manipulate the contents of the fame buffer directly. There is need to go through the pipelines. Two approaches to go:

- Bit-block-transfer(advanced ch.8)
- Logic operations (simple)

Rubber-banding: drawing a line segment by clicking to define the stating point then drag to locate the end points. During the rubber-banding we need to draw the line, remove it and restore the screen contents, redraw the line and son

Logic operations

- Writing in the frame buffer(the new pixel vale is called the source and the old pixel value is called the destination value)
- There are other 16 way for getting the new destination value from the source and the current destination values (only two will be discussed now)
  - Copy/replacement mode (default/normal rendering): New pixel values replaces the old values (source pixel replaces the destination pixel)
  - XOR: Source bits are XORed with current destination bits to get the new bits
    - Important property: (d XOR s) XOR s=d: Draw the object then erasing it and restoring it place by just redrawing it (a simple solution for rubber-banding and menu problem)